ANL/mcs/CP--87415
CONF-9509285--1

Additive Synthesis with DIASS_M4C on Argonne National Laboratory's IBM POWERparallel System (SP)

Hans Kaper
Argonne National Laboratory
kaper@mcs.anl.gov

David Ralley
Computer Music Project
Uniersity of Illinois
d-ralley@uiuc.edu

Juan Restrepo
Argonne National Laboratory
restrepo@mcs.anl.gov

Sever Tipei
Computer Music Project
University of Illinois
s-tipei@uiuc.edu

ABSTRACT: DIASS_M4C, a digital additive instrument, was implemented on the Argonne National Laboratory's IBM POWERparallel System (SP). This paper discusses the need for a massivley parallel supercomputer and shows how the code was parallelized. The resulting sounds and the degree of control the user can have justify the effort and the use of such a large computer.

DIASS, a Digital Instrument for Additive Synthesis on Supercomputers, is designed to allow control over the finest details in sound synthesis and to uncover the possibilities that current supercomputer architectures can offer the experimental music composer. In **DIASS**, any sound can have up to 65 partials (sine waves), but this current limit can be replaced by any arbitrary number (Kriese, C. and Tipei, S). Each wave/partial is defined by 12 static parameters or constant values (such as start time, duration, hall size and decay time for the reverberation) and 13 dynamic parameters or time-variant values (such as pitch, amplitude, tremolo, vibrato, transients, mix between direct and reverberated sound). The dynamic parameters use macros, or envelopes, of up to ten segments to define their behavior in time; these envelopes are custom designed by the user/composer, who can specify the *x* or time values, the *y* values and a path to be followed between *y* values. **DIASS** functions within the framework of the **M4C** sound synthesis language, an expanded C version of the **4BF** program from the *MusicN* family.

At 65 partials/sound and 25 controlling parameters for each partial, the amount of data to be specified by the user soon becomes hard to manage. **Scor5**, an editor or scorewriter, assists the user in creating an ASCII score file, a stack of "I cards" each 41 lines long. Through an interactive menu, the user creates every sound from scratch and later can edit it. A number of macros help increase the efficiency of the process by automatizing certain operations: assigning amplitudes to all partials of a sound when given particular envelopes and either a percentage of the fundamental's amplitude or a "taper" factor; ensuring that when the user specifies a certain loudness (in sones), the same perceived level results for sounds of different frequencies and number of partials (equal loudness); calculating the correct frequencies for all partials so they stay in the same relationship during a glissando; coordinating the use of randomness during vibrato or tremolo so that all partials of a sound are affected in the same way, at the same time and "gel" into a complex timbre; making sure that at no time the ceiling of maximum amplitude is crossed. Even then, choosing this many options for each of the hundreds of sounds that make up a piece is tedious. Since the most efficient use of **DIASS** is in connection with a computer-assisted composition program, an interface, **diaint.f**, automatically translates the composition program's output into a script to be used as input for **Scor5**.

The time and memory requirements for **DIASS_M4C** can vary according to the complexity of each sound and of the music in general. The more partials (waves) in a sound, the more options selected, or the more simultaneous sounds, the longer it takes to compute a second of music. To give an example, apiece for three voices (streams of sounds and/or chords), lasting about 12 minutes and containing 674 individual sounds some with 21 waves, some with 34, and some with 65, generated at a sampling rate of 22,050 Hz, stereo,

involves a 60 MB sound file and a 30 MB score file. On an IBM RS/6000, it takes about 90 min. to write the score with **Scor5** from the script produced with **diaint.f** and about 12 times longer to compute the samples.

It is obvious that even such a relatively modest piece requires a high-speed computer with large amounts of RAM and disk space. **DIASS** was developed successively on NeXT, CRAY Y-MP, and RS/6000 platforms, but none of them appeared adequate for the production stage. Only a massively parallel computer such as Argonne National Laboratory's **IBM POWERparallel System (SP)**, a cluster of 128 RS/6000s with 128 MB RAM on each node, could become an effective tool for the composer.

So far we concentrated on modifying the code for **DIASS_M4C**, the most time-consuming part of the process, in order to enable it to run in parallel on the **IBM SP**. Our approach is that of "task farming": individual sounds are computed on different nodes and the results are mixed together at the end. One node controls the entire operation and assigns tasks to other nodes as they become available. The user provides the score file (list of sound events to be synthesized) and requests a number of nodes on which the computations are to take place.

The decision of parallelizing at the sound level and not at a lower one was made for two main reasons: (1) the efficiency of message passing between nodes was estimated to be at its peak at this level; and (2) within a sound, individual partials need to have access to the same random numbers in order to coordinate non-deterministic features such as vibrato, transients and tremolo. The tool used was **MPI**, a portable parallel programming message-passing interface and library (Gropp, W., Lusk, E., Skjellum, A).

Preliminary results were convincing: a 6 minute piece of 200 sounds 16 waves each was computed in a little less than 1 hour, showing that the time for generating the binary samples is almost exactly the time necessary for sequential computation divided by the number of nodes used. It follows that with 100 or so nodes, the 12 minute piece described above could be computed in less than real time.

Additive synthesis is an expensive proposition, and **DIASS_M4C** takes it almost to the limit of what supercomputer architectures can offer at the present time. Why then not settle for alternatives that are less costly and easier to implement ? The answer is in the kind of sounds which would be very hard, if not impossible, to obtain any other way: sounds whose harmonics are progressively "detuned", to produce percussion- or noise-like timbres, and noises that are transformed into sounds through tuning; the "morphing" of a complex timbre which is decomposed into chords or, even further, into individual sine waves; as well as the reverse process; the use of amplitude and frequency transients to better approximate acoustic instruments when used in moderation or to create unusual sonorities when exaggerated; and the accurate scaling of amplitudes, taking into account masking effects and the Fletcher-Munson curves of equal loudness to reflect the loudness level desired by the composer, even in the case of large chords made out of complex timbres.

More than anything else, though, on Argonne National Laboratory's **IBM SP** supercomputer, **DIASS_M4C** is a practical instrument that gives the user control over the internal structure of a sound in as much detail as desired and as far as his/her imagination and patience will go. It is easy to see also how **DIASS_M4C** might become a powerful tool for the sonification of complex scientific data.

References: -Gropp,W.,Lusk,E. and Skjellum, A. - ***Using MPI***, The MIT Press, 1994.

-Kriese, C., and Tipei, S. - ***A Compositional Approach to Additive Synthesis on Supercomputers***, Proceedings of the 1992 International Computer Music Conference, San Jose, California, 1992.

Acknowledgments: This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under contract W-31-109-Eng-38. Part of the research was funded by Univ. of Illinois Research Board. Thanks to D. Blumenthal, M. Lauria, T. Lawrence, and S. Pakin for writing the parallel version of M4C.

DISCLAIMER